VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

BAKALÁŘSKÁ PRÁCE

2009 Radek Pindora

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Implementace skyline systému ukládání matic do knihovny OOSol

Implementation of Skyline Sparse Matrix Storage System to OOSol Library

2009

Radek Pindora

Poděkování

Tímto bych chtěl poděkovat panu Doc. Mgr. Vítu Vondrákovi, Ph.D. za pomoc a vedení při psaní této práce.

Prohlášení o autorství

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 7.5.2009 Podpis:

Abstrakt

Tato bakalářská práce seznamuje čtenáře se skyline systémem ukládání řídkých matic, jenž byl vytvořen pro řídké matice s převahou prvků ležících u diagonály. Obsahuje základní algoritmy pro operace se skyline maticemi a následně jejich konkrétní implementaci do knihovny OOSol, vyvíjené na Katedře aplikované matematiky FEI VŠB-TUO.

Dále jsou zde zpracovány BLAS (Basic Linear Algebra Subprograms) rutiny druhé úrovně, optimalizované pro práci se skyline maticemi. Implementace odpovídá definicím rutin v dokumentu An Updated Set of Basic Linear Algebra Subprograms [4].

Klíčová slova

Skyline matice, Skyline systém ukládání dat, řídká matice, BLAS, C++

Abstract

This bachelor thesis introduces the reader to the skyline sparse matrix storage scheme, which was made for sparse matrices with majority of elements lying near the diagonal. It includes basic algorithms for operations with skyline matrices and its implementation in the OOSol library, developed by the Department of Applied Mathematics, Faculty of Electrical Engineering and Computer Science, VŠB -Technical University of Ostrava.

Then level two BLAS routines (Basic Linear Algebra Subprograms) optimized for skyline matrices are elaborated. Implementation corresponds to definition in An Updated Set of Basic Linear Algebra Subprograms [4] document.

Keywords

Skyline matrix, Skyline storage format, sparse matrix, BLAS, C++

Obsah

Kapitola 1

Úvod

Vědecká práce a její promítnutí do praxe se často setkává s problémy při zpracovávání rozměrných matic, které jsou stále nad možnosti dnešních výpočetních technologií. V těchto momentech přicházejí na řadu specializované metody zpracovávání. Matice jsou studovány a roztříděny do tříd, podle jejich charakteristických vlastností. A jednou z těch tříd jsou řídké matice, respektive jejich speciální případ skyline matice.

Problém definování a zpracovávání skyline matic je obsahem této práce. Bude nastíněn teoretický postup při vytváření skyline sytému ukládání řídkých matic a jeho praktické využití v rámci lineárních operacích, takzvaných BLAS rutin. Toto vše je podpořeno ukázkami implementace v jazyku C++.

Hlavním cílem této bakalářské práce pak bude praktická implementace skyline systému do knihovny OOSol [10], která se vyvíjí na Katedře aplikované matematiky Vysoké školy báňské. Dále budou uvedeny příslušné BLAS (Basic Linear Algebra Subprograms) rutiny optimalizované pro práci s tímto konkrétním maticovým systémem. I přesto, že obecná implementace BLAS rutin je volně k dipozici, bylo výhodné vytvořit vlastní nezávislou úpravu. Její tvorba byla především zaměřená na využívání vlastností matic se skyline profilem a tedy větší efektivnost. Nezbytnou součástí práce je ukázka použití vytvořených metod, včetně ověřování a testování výsledků operací v programu Matlab.

Kapitola 2

Definice pojmů

Skalár x je reálné číslo.

Vektorem $\vec{v}$ rozumíme uspořádanou n-tici prvků – obvykle skalárů. Vektor může být sloupcový, nebo řádkový:

$$\vec{v} = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}$$

Matice A je obdélníková tabulka čísel či jiných objektů složena z m řádků a n sloupců – matice typu $m \times n$:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & & & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

Prvek matice je objekt obsažený v matici. Je určen svou souřadnicí: $[A]_{m,n}$

$$[A]_{1,1} = a_{11}$$

O nulovém prvku hovoříme v případě, že $[A]_{m,n} = 0$, opakem je nenulový prvek.

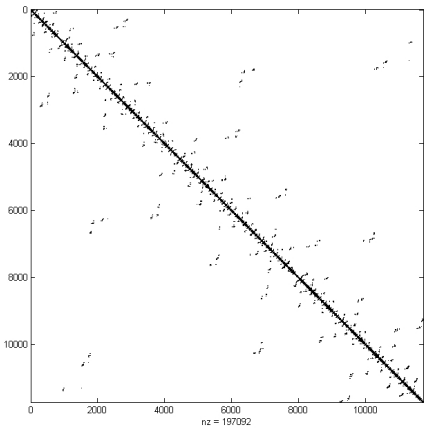
Transponovaná matice A^T vznikne vzájemnou výměnou řádků a sloupců z původní matice. Pro její prvky platí $a^T_{n,m} = a_{m,n}$

$$A^T = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ a_{13} & a_{23} & \cdots & a_{m3} \\ \vdots & & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix}$$

Čtvercová matice je matice, jejíž rozměr je $m \times m$

Symetrická matice je speciálním případem čtvercové matice, pro kterou platí $A^T = A$

Řídké matice je speciální případ obecné matice, který obsahuje výrazně více nulových prvků, než prvků nenulových. Ukázka je na obrázku 2.1 (nenulové prvky jsou zobrazeny černě). Obrázek byl vytvořen příkazem *spy*() v aplikaci MATLAB.



Obrázek 2.1: Ukázka řídké matice

Horní trojúhelníková část je část matice nalézající se NAD diagonálou matice. Pro její prvky $[A]_{m,n}$ platí, že $n > m$.

Dolní trojúhelníková část je část matice nalézající se POD diagonálou matice. Pro její prvky $[A]_{m,n}$ platí, že $n < m$.

Řádkem matice rozumíme vektor $r^A(m)$ obsažený v matici A

$$r^A(1) = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \end{bmatrix}$$

Sloupcem matice rozumíme vektor $s^A(n)$ obsažený v matici A

$$s^A(n) = \begin{bmatrix} a_{1n} \\ a_{2n} \\ \vdots \\ a_{mn} \end{bmatrix}$$

Diagonála matice je tvořena prvky $a_{11}, a_{22}, ..., a_{dd}$, kde $d = min(m, n)$

$$diag(A) = \begin{bmatrix} a_{11} & a_{22} & \cdots & a_{dd} \end{bmatrix}$$

Kapitola 3

Skyline systém ukládání matic

3.1 Úvod do skyline systému

Skyline matice jsou speciálním případem řídkých matic, jejíž prvky (nenulové) jsou přednostně uloženy na diagonále, případně v její blízkosti. Její profil se vyznačuje specifickým tvarem připomínající siluetu města s mrakodrapy [9] a proto se tato vlastnost využívá k efektivnější zpracovávání těchto matic. skyline matice jsou často využívány pro metodu konečných prvků, protože svůj profil zachovávají při Choleského dekompozici [5, 12] případně při Gaussově eliminaci [3].

Ukázka matice S se skyline profilem:

$$S = \begin{bmatrix} 0 & 1 & & 3 & & 5 & & & & \\ & 11 & & 13 & & 15 & & 17 & & \\ 20 & 21 & 22 & 23 & & 25 & & 27 & & \\ & & & & 34 & 35 & & 37 & & 39 \\ & 41 & 42 & 43 & 44 & 45 & 46 & 47 & & 49 \\ & & & 53 & 54 & 55 & 56 & 57 & & 59 \\ 60 & 61 & 62 & 63 & 64 & 65 & 66 & 67 & & 69 \\ & & & & & 75 & 76 & 77 & 78 & 79 \end{bmatrix}$$

Takovéto matice je výhodné (pro zmenšení paměťové náročnosti) ukládat do jednorozměrných vektorů specifickým způsobem, při kterém se bere v potaz jejich profil. Případně je zohledněna symetričnost.

3.2 Analýza skyline matic

Skyline matice jsou lehce identifikovatelné pomocí svého typického profilu. Prvky nad diagonálou (horní trojúhelníková část) tvoří sloupcové vektory s počátkem na diagonále a koncem u prvního nenulového prvku obsaženého v tomto sloupci. Prvky pod diagonálou (dolní trojúhelníková část) analogicky tvoří řádkové vektory s počátkem na diagonále a koncem u prvního nenulového prvku v tomto řádku. Velikost těchto vektorů se stává sloupcovým profilem, resp. řádkovým profilem skyline matice.

Pro názornost konkrétní příklad:

$$A = \begin{bmatrix} 11 & 12 & 13 & & & 16 \\ 21 & 22 & & 24 & & \\ & & 33 & 34 & & 36 \\ 41 & 42 & & 44 & & \\ & & & & 55 & 56 \\ & & & 64 & 65 & \end{bmatrix}$$

Prvky na diagonále jsou identifikovány jako:

$$diag(A) = \begin{bmatrix} 11 & 22 & 33 & 44 & 55 & 0 \end{bmatrix}$$

3.2.1 Sloupcový profil matice

Sloupcový profil matice udává velikost vektoru nad diagonálou, obsahující všechny nenulové prvky v tomto sloupci. Profil můžeme definovat [9] jako:

$$p_r(j) = j - min\{i : a_{ij} \neq 0, 1 \leq i \leq j\}$$

Nyní identifikujeme příslušný profil matice pro horní trojúhelníkovou část (pro názornost byly doplněné některé nulové prvky):

$$U = \begin{bmatrix} 11 & 12 & 13 & & & 16 \\ & 22 & 0 & 24 & & 0 \\ & & 33 & 34 & & 36 \\ & & & 44 & & 0 \\ & & & & 55 & 56 \\ & & & & & 0 \end{bmatrix}$$

Příslušné sloupcové vektory jsou:

$$\vec{s}_1 = \begin{bmatrix} 11 \end{bmatrix},\ \vec{s}_2 = \begin{bmatrix} 12 \\ 22 \end{bmatrix},\ \vec{s}_3 = \begin{bmatrix} 13 \\ 0 \\ 33 \end{bmatrix},\ \vec{s}_4 = \begin{bmatrix} 24 \\ 34 \\ 44 \end{bmatrix},\ \vec{s}_5 = \begin{bmatrix} 55 \end{bmatrix},\ \vec{s}_6 = \begin{bmatrix} 16 \\ 0 \\ 36 \\ 0 \\ 56 \\ 0 \end{bmatrix}$$

Všechny sloupcové profily matice uložené ve vektoru $\vec{p}_S$ jsou určeny velikostí výše uvedených sloupcových vektorů $\vec{s}_1$, ...,$\vec{s}_6$:

$$\vec{p}_S = \begin{bmatrix} 1 & 2 & 3 & 3 & 1 & 6 \end{bmatrix}$$

3.2.2 Řádkový profil matice

Řádkový profil matice udává velikost vektoru vlevo od diagonály, obsahující všechny nenulové prvky v tomto řádku. Profil můžeme definovat [9] jako:

$$p_r(i) = i - min\{j : a_{ij} \neq 0, 1 \leq j \leq i\}$$

Analogicky k tomu zpracujeme dolní trojúhelníkovou část matice (prvky na diagonále byly nahrazeny tečkami) A:

$$L = \begin{bmatrix} \cdot & & & & & \\ 21 & \cdot & & & & \\ & & \cdot & & & \\ 41 & 42 & 0 & \cdot & & \\ & & & & \cdot & \\ & & & 64 & 65 & \cdot \end{bmatrix}$$

Příslušné řádkové vektory jsou:

$$\vec{r}_1 = \begin{bmatrix} \emptyset \end{bmatrix}$$

$$\vec{r}_2 = \begin{bmatrix} 21 \end{bmatrix}$$

$$\vec{r}_3 = \begin{bmatrix} \emptyset \end{bmatrix}$$

$$\vec{r}_4 = \begin{bmatrix} 0 & 42 & 41 \end{bmatrix}$$

$$\vec{r}_5 = \begin{bmatrix} \emptyset \end{bmatrix}$$

$$\vec{r}_6 = \begin{bmatrix} 65 & 64 \end{bmatrix}$$

Kompletní řádkový profil matice tedy je:

$$\vec{p}_R = \begin{bmatrix} 0 & 1 & 0 & 3 & 0 & 2 \end{bmatrix}$$

V tomto názorném rozdělení matice na příslušné vektory již lze vidět podstatu práce se skyline maticemi a jejích zpracovávání na výpočetní technice. Při vhodně tvarované matici můžeme dosáhnout podstatné úspory paměťových nároků na uložení, oproti postupnému výpisu prvků v případě řídkých matic nebo dokonce hustých matic.

3.3 Ukládání skyline matic

Pro obecnou řídkou matici S skyline systém ukládání vytváří 4 vektory [6]: $\overrightarrow{uc}$, $\overrightarrow{lr}$, $\overrightarrow{iuc}$, $\overrightarrow{ilr}$. Vektory $\overrightarrow{uc}$ a $\overrightarrow{iuc}$ náleží horní trojúhelníkové části. Vektory $\overrightarrow{lr}$ a $\overrightarrow{ilr}$ spodní trojúhelníkové část. Případně se může vytvořit ještě další vektor $\overrightarrow{diag}$ pro diagonální prvky matice.
Teoretický postup zpracovávání matice byl již uveden výše. Dalším krokem nastává vytvoření vhodných jednorozměrných polí – vektorů $\overrightarrow{uc}$, $\overrightarrow{lr}$, $\overrightarrow{iuc}$, $\overrightarrow{ilr}$ a jejich naplnění prvky [6].

3.3.1 Popis vektorů

$\overrightarrow{uc}$ – obsahuje prvky horní trojúhelníkové matice, konkrétně výše uvedené transponované sloupcové vektory $\vec{s}_1 \ldots \vec{s}_n$ seřazené za sebou. Její rozměr je dán sumou všech sloupcových profilů $\left|\overrightarrow{uc}\right| = \sum_{k=1}^{n} s_k$. Jsou v ní obsaženy i nulové prvky, obsahuje-li je původní profil matice.

$$\overrightarrow{uc} = \begin{bmatrix} \vec{s}_1^T & \vec{s}_2^T & \cdots & \vec{s}_n^T \end{bmatrix}$$

$\overrightarrow{lr}$ – obsahuje prvky dolní trojúhelníkové matice – výše uvedené řádkové profily $\vec{r}_1 \ldots \vec{r}_m$ seřazené za sebou. Její rozměr je dán sumou všech řádkových profilů $\left|\overrightarrow{lr}\right| = \sum_{k=1}^{m} r_k$. Taktéž jsou v ní zahrnuty i nulové prvky.

$$\overrightarrow{lr} = \begin{bmatrix} \vec{r}_1 & \vec{r}_2 & \cdots & \vec{r}_m \end{bmatrix}$$

$\overrightarrow{\boldsymbol{iuc}}$ – pole čísel obsahující relativní souřadnice počátků všech sloupcových vektorů obsažených v $\overrightarrow{uc}$. Jeho velikost je $n + 1$.

$\overrightarrow{\boldsymbol{ilr}}$ – pole čísel obsahující relativní souřadnice počátků všech řádkových vektorů obsažených v $\overrightarrow{lr}$. Jeho velikost je $m + 1$.

$\overrightarrow{\boldsymbol{diag}}$ – pole obsahující všechny diagonální prvky matic. Jeho velikost je $\left|\overrightarrow{diag}\right| = min(m, n)$. Nemusí být nutně obsaženo v systému, diagonální prvky jsou zpravidla uloženy v $\overrightarrow{uc}$.

3.3.2 Vytváření skyline systému

Vytvoření pole $\overrightarrow{uc}$, lze popsat jako postupné vkládání transponovaných sloupcových vektorů $\vec{s}_1 \ldots \vec{s}_n$:

$$\overrightarrow{uc} = [11\ 22\ 12\ 33\ 0\ 13\ 44\ 34\ 24\ 55\ 0\ 56\ 0\ 36\ 0\ 16]$$

Do pole $\overrightarrow{iuc}$ jsou uloženy relativní souřadnice začátků vektorů v poli a jeho poslední prvek $\overrightarrow{uc}$:

$$\overrightarrow{iuc} = [1\ 2\ 4\ 7\ 10\ 11\ 16]$$

Analogicky je toto provedeno pro dolní trojúhelníkovou matici ($*$ značí diagonální prvky, které není nutné znovu vkládat, nicméně je potřeba vytvořit pozici [6]):

$$\overrightarrow{lr} = [*\ *\ 21\ *\ *\ 0\ 42\ 41\ *\ *\ 65\ 64]$$

$$\overrightarrow{ilr} = [1\ 2\ 4\ 5\ 9\ 10\ 13]$$

Nyní jsou definovány a korektně uloženy všechny nenulové prvky matice A ve čtyřech jednorozměrných polích (vektorech). $\overrightarrow{iuc}$ a $\overrightarrow{ilr}$ jsou pouze celočíselnými ukazateli, mají tedy malou paměťovou náročnost. Pro samotné hodnoty matice v $\overrightarrow{uc}$ a $\overrightarrow{lr}$ je potřeba zvolit vhodnou přesnost (datový typ), aby se nezvyšovaly zbytečně paměťové nároky.

3.3.3 Přístup k uloženým prvkům

Samotný přístup k prvkům je lineárním problémem. Je potřeba zkontrolovat pozici požadovaného prvku x. Možnosti jsou tři: nad diagonálou, na diagonále a pod diagonálou. Dle toho

je určeno pole, ve kterém se prvek nachází. Dále je potřeba ověřit zda se prvek nalézá v definovaném profilu matice, v negativním případě se $x = 0$. Je-li ověřeno, že prvek je skutečně uložen v $\overrightarrow{uc}$ nebo $\overrightarrow{lr}$, pak následuje jeho dohledání pomocí jednoduchého přístupu k poli na základě pozice určené v $\overrightarrow{iuc}$ nebo $\overrightarrow{ilr}$.

Jednoduchý algoritmus vykonávající tyto operace:

```
double get(col,row)
{
    if(row <= col) //nad diagonalou
    {
        profil = col - row; //vzdalenost od diagonaly
        if( profil > colProfil(col) ) return 0; //prvek nelezi ve sloupcovem profilu
        return uc[ iuc[col] + profil ]; //vyhledani hodnoty v poli uc
        }
    if(row > col) //pod diagonalou
    {
        profil = row - col; //vzdalenost od diagonaly
        if( profil > rowProfil(row) ) return 0; //prvek nelezi v radkovem profilu
        return lr[ ilr[row] + profil ]; //vyhledani hodnoty v poli lr
    }
}
```

3.3.4 Vkládání nových prvků

Pro časté přidávání prvků do matice není tento systém navržen. Vzhledem ke statickému uložení vektorů, je někdy nezbytné provést novou alokaci některého z pole $\overrightarrow{uc}$ nebo $\overrightarrow{lr}$, a to v případě, kdy je nutné zvětšit řádkový nebo sloupcový profil matice.

V praktické implementaci je při vkládání dalšího prvku do matice nutné nejdříve ověřit, zda leží v profilu matice, nebo ne. V prvním případě (leží v již definovaném profilu), dochází k jednoduchému nastavení prvku v poli $\boldsymbol{\overrightarrow{uc}}$ pro horní trojúhelníkovou matici, případně $\boldsymbol{\overrightarrow{lr}}$ pro dolní trojúhelníkovou matici. V opačném případě dochází k paměťově náročné realokaci pole $\overrightarrow{uc}$ nebo $\overrightarrow{lr}$. Velikost pole je inkrementována o rozdíl ve velikosti příslušného profilu matice a následně dojde k vhodnému překopírování hodnot z původního pole, pak je původní pole smazáno z paměti. Nyní je možné úspěšně vložit zadaný prvek na jeho pozici.

3.4 Implementace pro OOSol

Konkrétní implementace, zde popsána, je malou úpravou obecného systému pro ukládání skyline matic [6]. Jednotlivé implementace se můžou lišit podle konkrétní situace, podstata (práce s řádkovým a sloupcovým profilem matice) ovšem zůstává stejná.

Například jedna ze zajímavých implementací byla vytvořená pro Intel MKL (Intel ©Math Kernel Library) [7] – pracuje pouze se dvěma vektory (hodnoty a ukazatele) a je tedy schopna pracovat pouze s trojúhelníkovou maticí. Intel MKL je implementací náročných výpočetních rutin, které jsou optimalizovány pro použití na stanicích s procesory Intel. Mimo jiné obsahuje BLAS rutiny, Fourierovy transformace, řídké řešiče a řešení funkcí s vektorovým argumentem.

3.4.1 Popis úprav

Výše uvedený přístup tvorby skyline systému ukládání matic zůstal zachován, kromě těchto změn:

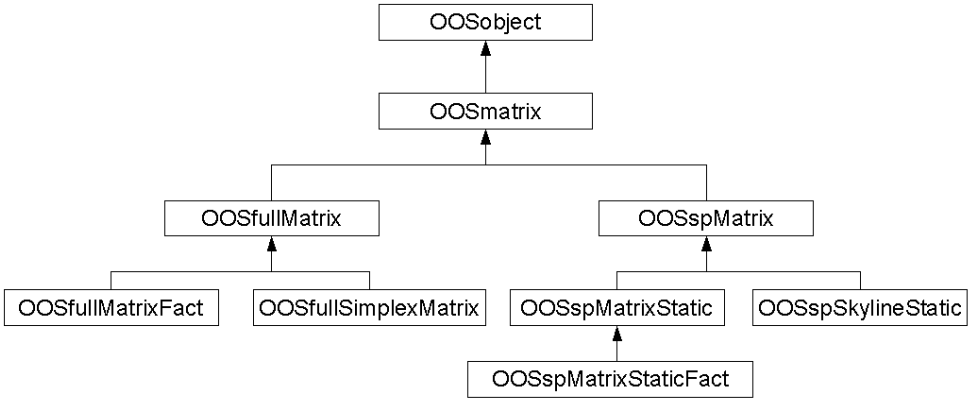
- Bylo vytvořeno oddělené pole pro prvky na diagonále – $\overrightarrow{diag}$.
- Ve vektoru $\overrightarrow{\boldsymbol{uc}}$ nejsou obsaženy diagonální prvky.
- Ve vektoru $\overrightarrow{\boldsymbol{lr}}$ nejsou vyhrazená místa pro diagonální prvky, původně zastoupená $*$.
- Ve vektorech $\overrightarrow{\boldsymbol{iuc}}$ a $\overrightarrow{\boldsymbol{ilr}}$ se sloupce resp. řádky s nulovým profilem označují hodnotou minulého sloupce, resp. řádku.
- Vzhledem k malé paměťové náročnosti jsou uchovávány navíc i vektory s řádkovými a sloupcovými profily.

Tyto úpravy originálního skyline systému ukládání napomohly k jednodušší a efektivnější implementaci skyline systému ukládání matic. Především se optimalizovaly operace pro přístup k prvkům ($get()$ a $set()$), díky jednoduššímu vyhledávání v poli $\overrightarrow{\boldsymbol{iuc}}$ a $\overrightarrow{\boldsymbol{ilr}}$. Také došlo k urychlení operace $get()$ pro snadné ověření, zda leží požadovaný prvek v definovaném profilu matice. Stejně tak jako přístup k diagonálním prvkům, pro které je nyní vyhrazeno zvláštní pole.

3.4.2 OOSol

Skyline systém ukládání řídkých matic byl implementován v třídě *OOSspSkylineStatic*. Jako předek posloužila vhodná abstraktní třída *OOSspMatrix*. Celá implementace souborů *OOSspSkylineStatic.cpp* a *OOSspSkylineStatic.h* je přiložená v příloze k této práci.

Na obrázku 3.1 je zobrazen diagram tříd obsažených OOSol. Pro přehlednost jsou zahrnuty pouze třídy dědící z *OOSmatrix*.



Obrázek 3.1: Diagram tříd OOSol

3.4.3 Implementované metody

K zajištění funkčnosti skyline systému ukládání řídkých matic bylo nutné vytvořit následující metody. Při implementaci bylo vycházeno z již existující [9] třídy, optimalizované pro Sloanův algoritmus. Vzhledem k odlišnému cíli (implementace BLAS rutin) této práce, byla vytvořena vlastní úprava algoritmu, se snahou o zachování původní struktury kódu. Ve třídě *OOSspSkylineStatic* jsou vytvořeny mimo jiné následující metody:

- *OOSspSkylineStatic*() konstruktor alokující paměť pro základní datové struktury.
- *OOSspSkylineStatic*($TIND\ row$, $TIND\ col$) konstruktor alokující paměť pro obdélníkovou matici.

- $OOSspSkylineStatic(TIND\ size)$ konstruktor alokující paměť pro čtvercovou matici.
- $OOSspSkylineStatic(TIND\ *row,\ TIND*col,\ double\ *val)$ konstruktor vytvářející skyline matici ze 3 vektorů. Každý nenulový prvek je uložen ve vektoru $\overrightarrow{val}$ a jeho souřadnice jsou uvedeny v $\overrightarrow{col}$ a $\overrightarrow{row}$.
- $OOSspSkylineStatic()$ destruktor třídy – maže všechny třídní prvky (používá metodu $del()$.
- $void\ clean()$ metoda vymazání všech třídních prvků.
- $void\ init()$ vymaže a znovu alokuje všechny třídní prvky.
- $void\ skylineConvert(TIND\ *prvkyRow,\ TIND\ *prvkyCol,\ double\ *prvky)$ metoda obstarávající samotné vytváření skyline matice.
- $void\ trans()$ transponování matice. Toto je provedeno jednoduchým prohozením vektorů $\overrightarrow{uc}$ a $\overrightarrow{lr}$, $\overrightarrow{iuc}$ a $\overrightarrow{ilr}$.
- $void\ resize(TIND\ dim)$ alokace matice s novým čtvercovým rozměrem dim.
- $void\ resize(TIND\ nrow,\ TIND\ ncol)$ alokace nové obdélníkové matice $nrow \times ncol$.
- $TIND\ getCols()$ navrátí počet sloupců matice.
- $TIND\ getRows()$ navrátí počet řádků matice.
- $TIND\ nnz()$ navrátí celkový počet nenulových prvků.
- $TIND\ getDiagonal(TIND\ i)$ navrátí diagonální prvek na pozici $[i, i]$
- $void\ setIOFormat(TIND\ i)$ nastaví vstupně/výstupní formát dat. 1 pro řídké matice, 0 pro skyline matice.
- $TIND\ columnHeight(TIND\ col)$ vrátí sloupcový profil na pozici col.
- $TIND\ rowHeight(TIND\ row)$ navrátí řádkový profil na pozici row
- $double\ get(TIND\ row,\ TIND\ col)$ navrátí prvek na pozici $[col, row]$
- $void\ set(TIND\ col,\ TIND\ row,\ double\ val)$ nastaví prvek na pozici $[col, row]$ na hodnotu danou proměnnou val. V případě, že prvek leží mimo existující profil matice, provede realokaci nové matice.
- $OOSspMatrixStatic*\ toDenseMatrix()$ překonvertuje skyline matici na třídu OOSsp-MatrixStatic.

- *ostream& operator << (ostream&os, OOSspSkylineStatic& mat)* vypíše matici do souboru.
- *ifstream& operator >> (ifstream&is, OOSspSkylineStatic& mat)* načte matici ze souboru.
- *ofstream& operator << (ofstream&os, OOSspSkylineStatic& mat)* na konzoli vypíše testovací zprávu o matici.

3.5 Aplikace třídy OOSspSkylineStatic

3.5.1 Vstupní soubor

Implementováno je vytváření matice ze souboru ve formátu pro ukládání řídkých matic. Tento soubor má následující formát:

```
6 6 17
1 1 11
1 2 12
1 3 13
2 1 21
2 2 22
2 4 24
3 3 33
3 4 34
3 6 36
4 1 41
4 2 42
4 4 44
5 5 55
5 6 56
6 4 64
6 5 65
1 6 16
```

Ukázka zápisu matice A definované v kapitole 3.2.

První tři uspořádané prvky označují: n, m, počet prvků matice. Dále jsou vypsány všechny obsažené prvky v matici: *dek*, *sloupec*, *hodnota*. Nevýhodou použití tohoto formátu je nutnost jeho předzpracování a upravení pro uložení do implementovaného skyline systému. Práce se soubory je přizpůsobená použití standardních streamu << a >>.

3.5.2 Aplikace třídy OOSspSkylineStatic

Následující zdrojový kód ukazuje vytváření a práci s třídou *OOSspSkylineStatic* (jedná se o testovací soubor *OOStest.cpp*):

```
#include "OOSvectors.h"
#include "OOSmatrices.h"
#include "OOSspSkylineStatic.h"
#include <iostream>
#pragma comment(lib,"oosol.lib")

using std::cout;
using std::cin;
using namespace oosol;

int main()
{
    try
    {
        cout << "Test OOSspSkylineStatic\n";
        OOSspSkylineStatic *S = new OOSspSkylineStatic();
        S->setText();
        S->setIOFormat(1);

        ifstream f("A.txt");
        f >> *S;
        f.close();

        cout << *S;

        S->set(4,0,123);
        S->trans();
        S->setBinary();
        S->setIOFormat(1);
        ofstream o("fullMatrixOUT.txt");
        o << *S;
        o.close();

    } catch (OOSexception e)
    {
        e.print();
        cin.get();
        return 1;
    }
    cout << "\n\nEnd...";
    cin.get();
    return 0;
}
```

3.5.3 Výstup programu

Ukázka výstupu programu do konzole (pro uvedený vstupní soubor reprezentující výše uvedenou matici A):

```
Test OOSspSkylineStatic

Testovací výpis Matice:
Col: 6 Rows: 6 Elements: 17

 Sloupcový profil:  0 1 2 2 0 5
 Řádkový profil:    0 1 0 3 0 2

Prvky na diagonále:  11 22 33 44 55 0

iuc: -1 0 1 3 3 5 10
uc: 12 0 13 34 24 56 0 36 0 16

ilr: -1 0 0 1 1 4 6
lr: 21 0 42 41 65 64

Matice:
11      12      13      0       0       16
21      22      0       24      0       0
0       0       33      34      0       36
41      42      0       44      0       0
0       0       0       0       55      56
0       0       0       64      65      0
```

Kromě případného výstupu na konzolu může být i vytvořen výstupní soubor, ve kterém jsou uloženy všechny potřebné hodnoty a ze kterého může být matice opět načtena. Možnosti pro formátování výstupu jsou dvě, jedna byla již uvedená jako vstupní matice, další je speciálně navržená pro skyline matice. Volba se provádí pomocí metody $setIOFormat(TIND)$ Příklad vytvořeného souboru ($setIOFormat(0)$):

```
6 6 17
6 7 14 7
11 22 33 44 55 0
21 0 42 41 65 64
-1 0 1 3 3 5 10
21 0 42 41 65 64
-1 0 0 1 1 4 6
0 1 0 3 0 2
0 1 2 2 0 5
```

Tento výstupní soubor se drží následujícího formátování:

```
počet sloupců, počet řádků, počet prvků
velikost uc, velikost iuc, velikost lr, velikost ilr
diagonální prvky
prvky ve vektoru uc
prvky ve vektoru iuc
prvky ve vektoru lr
prvky ve vektoru ilr
řádkový profil matice colProfile
sloupcový profil matice rowProfile
```

Jeho výhodou je snadná interpretace hodnot v jednotlivých vektorech a jednoduché a nenáročné načtení do skyline systému ukládání matic. Odpadá nutnost výpočetně náročného analyzování matic a jejich následná transformace do podoby vhodné pro skyline systém. Nevýhodou se jeví nutnost ukládání kompletního profilu matice – u řídké matice vznikne objemnější soubor.

Kapitola 4

BLAS

4.1 Popis BLAS knihovny

BLAS [11] (Basic Linear Algebra Subprograms) jsou základní rutiny pro práci s vektory a maticemi. Obsahují pouze nezbytné operace s reálnými, nebo komplexními vektory a maticemi – nejsou zde tedy implementovány řešiče lineárních rovnic a maticové rozklady. Podstatou BLAS rutin je dosažení co největší možné efektivity, přenositelnosti a šíře použití. Proto jsou využívány v kvalitních aplikacích pro lineární algebru (LAPACK [2] například). Často jsou vytvářený optimalizace pro konkrétní výpočetní architekturu [8], případně je možné si vytvořit vlastní optimalizovanou knihovnu pomocí ATLASu – Automatically Tuned Linear Algebra Software [1]. Důvodem je snaha o efektivitu prováděných operací. Vzhledem k tomu, že BLAS obsahuje základní stavební prvky pro úlohy lineární algebry, lze předpokládat jeho časté a náročné využívání. Nezbytností je proto provádět optimalizace algoritmu implementovaných metod, pro zajištění maximální rychlosti a efektivnosti. Původní verze byla programována v jazyce Fortran77, ale nyní jsou k dispozici i jiné verze v jazyce C, nebo Java. V této implementaci je použit jazyk C++.

4.1.1 Struktura BLAS

BLAS se rozděluje na tyto úrovně, podle operandů:

Level 1 obsahuje skalární, vektorové a vektor-vektor operace.

$$y \leftarrow \alpha x + y$$

Level 2 obsahuje operace matice-vektor.

$$y \leftarrow \alpha A + \beta y$$

Level 3 obsahuje maticové operace.

$$C \leftarrow \alpha AB + \gamma C$$

Rozdělení BLAS podle přesnosti číselných operandů:

REAL reálný s jednoduchou přesností – float

DOUBLE PRECISION reálný s dvojitou přesností – double

COMPLEX komplexní s jednoduchou přesností – float*

DOUBLE COMPLEX komplexní s dvojitou přesností – double*

V OOSol knihovně se využívá datový typ double – reálné čísla s dvojitou přesností, který je podstatný i v implementaci skyline systému. Dále se budeme zabývat především implementací Level 2 BLAS rutin pro skyline matice.

4.2 OOSol a BLAS

OOSol je moderní knihovna se zaměřením na objekty a objektově orientovaný přístup, který by měl umožnit jednoduchou a přehlednou práci s implementovanými algoritmy, při řešení praktických problémů. V knihovně OOSol (Object Oriented Solvers) jsou také zahrnuty BLAS operace. BLAS rutiny v tomto případě nejsou pouhé funkce, které operují pouze s argumenty, ale celými třídami s vlastními metodami a objekty. Výhodou je přehlednost takovéhoto kódu a jednoduchost další implementace. Dále není nutné vždy odlišovat jednotlivé funkce pomocí prefixů – s použitím polymorfismu jazyka C++ se tento problém řeší na základě vstupních parametrů sám.

4.3 Implementace BLAS

Byla implementována 2. úroveň BLAS rutin pro použití se skyline systémem ukládání matic, tedy operace řídké matice a vektoru, jejich popis je uveden v další části práce. Dále byly implementovány maticové normy. Jedná se o optimalizovanou implementaci pro použití s *OOSspSkylineStatic* třídou. Optimalizace se dotkla především přístupu k prvkům matice – jednotlivé operace nevyužívající standardní *get*(), *set*() operace, ale přistupují přímo k definovaným polím s prvky.

4.4 Normy matic

Pro reálný typ matic jsou v BLAS definovány [4] tyto normy:

4.4.1 Sloupcová norma

$$||A||_1 = \max_j \sum_i |a_{ij}|$$

Ukázka implementace normy ve třídě *OOSspSkylineStaticBLAS.cpp*. Implementace normy byla prováděná pro skyline systém. Lze tedy vidět přímý přístup k třídním členům *uc* a *iuc*.

```
double OOSspSkylineStaticBLAS::n_one() //sloupcova
{
    TIND row = M->row, col = M->col;
    double max = 0;
    double *sum = new double[col];
    std::fill(sum,sum+col,0.0);
        for(TIND i = 0; i < M->size_diagonal; i++) //prvky na diagonale
    {
        sum[i] = ABS(M->diag[i]);
    }
    for(TIND c = 0; c < col; c++)
    {
        if(M->iuc[c] != -1) //prvky nad diagonalou
        {
            TIND u = M->iuc[c];
            TIND um = u + M->colProfile[c];
            for(TIND i = u; i < um; i++)
            {
                sum[c] += ABS(M->uc[i]);
            }
        }
        for(TIND i = c+1; i < row; i++) //prvky pod diagonalou
        {
            sum[c] += ABS(get(c,i));
        }
        }
    for(TIND i = 0; i < col; i++) max = MAX(max,sum[i]);
    return max;
    }
```

4.4.2 Řádková norma

$$||A||_\infty = \max_i \sum_j = |a_{ij}|$$

Implementace této normy byla analogická sloupcové normě, není zde proto uvedena. Pro podrobnosti je k nahlédnutí její komplení implementace v příloze.

4.4.3 Maximová norma

$$||A||_{max} = \max_i \max_j |a_{ij}|$$

Ukázka implementace normy ve třídě *OOSspSkylineStaticBLAS.cpp*:

```
double OOSspSkylineStaticBLAS::n_max()
{
    double max = get(0,0);
        for(TIND i = 0; i < M->size_AU; i++)
        max = MAX(max,ABS(M->uc[i]));
    for(TIND i = 0; i < M->size_AL; i++)
        max = MAX(max,ABS(M->lr[i]));
    for(TIND i = 0; i < M->size_diagonal; i++)
        max = MAX(max,ABS(M->diag[i]));
    return max;
}
```

4.4.4 Frobeniova norma

$$||A||_F = \sqrt{\sum_i \sum_j a_{ij}^2}$$

Ukázka implementace normy ve třídě *OOSspSkylineStaticBLAS.cpp*:

```
double OOSspSkylineStaticBLAS::n_frobenius()
{
    double max = 0;
        for(TIND i = 0; i < M->size_AU; i++)
        max += pow(M->uc[i],2);
    for(TIND i = 0; i < M->size_AL; i++)
        max += pow(M->lr[i],2);
    for(TIND i = 0; i < M->size_diagonal; i++)
        max += pow(M->diag[i],2);
    return sqrt(max);
}
```

4.5 BLAS Level 2

Zde následuje matematický popis a ukázka implementování BLAS rutin z druhé úrovně. Použité symboly:

$\boldsymbol{A}, \boldsymbol{B}$ skyline matice (OOSspSkylineStatic*)

$\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w}$ Vektory (OOSvectorD*)

$\boldsymbol{\alpha}, \boldsymbol{\beta}$ Skaláry (Double)

Pro skyline systém byly implementovány tyto operace:

MV

$$y \leftarrow \alpha Ax + \beta y$$

Implementace:

```
void mv(OOSvectorD *x, OOSvectorD *y, double alpha, double beta);
```

MTV

$$y \leftarrow \alpha A^T x + \beta y$$

Implementace:

```
void mtv(OOSvectorD *x, OOSvectorD *y, double alpha, double beta);
```

SUM_MV

$$y \leftarrow \alpha Ax + \beta By$$

Implementace:

```
void sum_mv(OOSspSkylineStatic *B, OOSvectorD *x, OOSvectorD *y,
     double alpha, double beta);
```

MVT

$$x \leftarrow \beta A^T y + z$$

$$w \leftarrow \alpha Ax$$

Implementace:

```
void mvt(OOSvectorD *z, OOSvectorD *y, OOSvectorD *w, OOSvectorD *x,
     double alpha, double beta);
```

MVER

$$A \leftarrow A + u_1 v_1^T + u_2 v_2^T$$

$$x \leftarrow \beta A^T y + z$$

$$w \leftarrow \alpha A x$$

Implementace:

```
void mver(OOSvectorD *z, OOSvectorD *y, OOSvectorD *x,
    OOSvectorD *w, OOSvectorD *u1, OOSvectorD *u2,
    OOSvectorD *v1, OOSvectorD *v2, double alpha,
    double beta);
```

R

$$A \leftarrow \alpha x y^T + \beta A$$

Implementace:

```
void r(OOSvectorD *x, OOSvectorD *y, double alpha, double beta);
```

R2

$$A \leftarrow (\alpha x) y^T + y(\alpha x)^T + \beta A$$

Implementace:

```
void syr2(OOSvectorD *x, OOSvectorD *y, double alpha, double beta);
```

4.6 Ověřování výsledků v MATLABU

Pro kontrolu výsledků násobení matic vektory a jiných operací, bylo nutné vytvořit rychlé a jednoduché metody pro použití s programem Matlab. K tomu bylo využito již implementované metody pro výstup matice do souborů, stejně tak i v případě vektorů. Takto vytvořené soubory byly načteny do Matlabu a zde postupně provedeny příslušné operace. Oba výsledky byly nakonec porovnány. Pro testování operací jsou v Matlabu nezbytné tyto příkazy:

load S.txt načte vektor, nebo matici z textového souboru *S.txt*

$\boldsymbol{A = spconvert(S)}$ převede načtený vektor na řídkou matici.

$\boldsymbol{o = A * x - y}$ ukázka operace násobení matice A vektorem x, od výsledného vektoru je ještě odečten vektor y.

$\boldsymbol{save\ O.txt\ O\ -ascii}$ uložení vzniklé matice O, případně vektoru do souboru $O.txt$ ve formátu Ascii.

Pro zjednodušení těchto stále opakujících se operací byl vytvořen testovací algoritmus, který definuje potřebné matice a vektory. Naplní je náhodnými prvky, a to vše uloží do příslušných souborů. Navíc je vytvořen i M-soubor, ve kterém jsou příslušné testovací operace pro Matlab. Zde přikládám testovací třídu:

```
std::cout.precision(10);

    //vytvoreni matic
OOSspSkylineStatic *A = new OOSspSkylineStatic(100);
OOSspSkylineStatic *B = new OOSspSkylineStatic(100);

    //naplneni matic nahodnymi cisly
for(TIND j = 0; j < A->getRows(); j++)
for(TIND i = j; i < A->getCols(); i++)
    {
        A->set(j,i,rand());
        B->set(j,i,rand());
    }

    //nasleduje testovani blas rutin
OOSvectorD *x = new OOSvectorD(A->getCols(),0.0);
OOSvectorD *y = new OOSvectorD(A->getCols(),0.0);
for(int i = 0; i < x->getAllocDim(); i++) x->set(i,i);
for(int i = 0; i < y->getAllocDim(); i++) y->set(i,i*i);
OOSvectorD *out = new OOSvectorD(*y);
double alpha = 1.0, beta = 1.0;

    //BLAS normy
cout << "\nNorm One: " << A->norm_1();
cout << "\nNorm Infinity: " << A->norm_inf();
cout << "\nNorm Max: " << A->norm_max();
cout << "\nNorm Frobenius: " << A->norm_F();

    //testovani operace sum_mv
A->sum_mv(B,x,out,alpha,beta);

    //ukazka vystupu do souboru test.m
ofstream m("E:\\out\\test.m");
m   << "function[] = test()" << endl
    <<"load A.txt; " << endl
    <<"A = spconvert(A);" << endl
    <<"load B.txt;" << endl
```

```
        <<"B = spconvert(B);" << endl
        <<"load X.txt;" << endl
        <<"load Y.txt;" << endl
        <<"load out.txt;" << endl
        <<"nA1 = norm(A,1)" << endl
        <<"nAi = norm(A,inf)" << endl
        <<"nAf = norm(A,'fro')" << endl
        <<"difference = norm(( A*X + B*Y ) - out,1)" << endl;

        //postupne vypisy objektu do souboru
    ofstream o("E:\\out\\A.txt");
    o << A->toString();
    o.close();
```

Výstupem tohoto programu jsou příslušné soubory pro matice a vektory. Navíc je vytvořen skript, pro snadné testování. Skript je spuštěn v Matlabu jednoduchým příkazem *test*. Ukázka takového skriptu:

```
function[] = test()
load A.txt;
A = spconvert(A);
load B.txt;
B = spconvert(B);
load X.txt;
load Y.txt;
load out.txt;
nA1 = norm(A,1)
nAi = norm(A,inf)
nAf = norm(A,'fro')
difference = norm(( A*X + B*Y ) - out,1)
```

Po vykonání skriptu se na obrazovce Matlabu zobrazí některé normy matice A. Je-li výsledek správný lze ověřit v proměnné $difference$. Tato proměnná je definována jako norma z vektoru rozdílů obou výsledků. V případě shodných výstupů z Matlabu i ze $OOSspSkylineStatic$ je jeho hodnota rovna 0 a výsledek je tedy považován za ověřený.

```
>> test
nA1 =
      146683

nAi =
      178497

nAf =
  1.3420e+005

difference =
     0
```

Výše lze vidět ukázkový výstup pro dvě náhodné matice (norma jenom pro A). Proměnná $difference = 0$ a proto lze tento příklad považovat za úspěšný. Touto jednoduchou metodou byla prováděna kontrola všech implementovaných BLAS operací. Pro důslednější ověření bylo generování náhodných matic opakováno několikrát pro různou definovanou velikost. Proměnná $difference$ nemusí být nezbytně nula, je třeba zahrnout možnost vzniku chyby vlivem zaokrouhlení. Pro zvýšení přesnosti je pak nezbytné upravit standardní výstup do souboru (případně konzoly) příkazem $precision(int)$.

Kapitola 5

Závěr

Podstatou této bakalářské práce byla vlastní implementaci skyline systému pro ukládání a zpracovávání řídkých matic i s příslušnými BLAS rutinami. Výsledkem toho byla vytvořena třída *OOSspSkylineStatic* implementována do knihovny OOSol. V minulosti [9], již byla třída *OOSspSkylineStatic* vytvořena pro použití se Sloanovým algoritmem pro snižování profilu matic. Implementace o které pojednává tato práce, byla vytvořena a optimalizovaná pro účel vytvoření příslušné BLAS knihovny s využitím u skyline systému ukládání matic. Její konstrukce byla nicméně upravena k obrazu třídy pana Stachoně, pojmenování metod a třídních prvků jsem se tedy snažil zachovat. Jednotlivé implementace metod jsou však již mnou vytvořené.

V rámci BLAS rutin bylo cílem nadefinovat 2. úroveň operací nad skyline systémem a implementovat jej do OOSol knihovny. Přiložené zdrojové kódy dokazují praktické vytvoření ve třídě *OOSspSkylineStaticBLAS* (viz. příloha) a odladění pro použití s třídou *OOSspSkylineStatic*. Tato třída je optimalizována pro přímý přístup k prvkům skyline matic – obchází tedy standardní *get*() a *set*(), které se stávají při použití skyline systému neefektivní.

I přes odvedenou práci zůstává skyline systém ukládání matic vhodný pro pokračující programátorský vývoj. Jako další krok se jeví využití jeho implementace pro faktorizaci matic a jejich následné řešení. Návrhy pro další postup:

Faktorizace matic vhodná pro práci se skyline systémem. Implementace LU rozkladu, Choleského rozkladu případně jiné metody vhodné pro použití se skyline maticí.

BLAS rutiny 3. úrovně pro práci se skyline maticemi.

Normy matice které nejsou definovány v BLAS.

Optimalizace případných implementovaných metod a především některých BLAS rutin. Prostorem se stávají operace, u kterých lze předpokládat vznik hustých matic (R, $R2$). Dále pak všechny metody používající k přístupu $get()$ (např. některé normy, vektorové operace) a $set()$ metody.

Literatura

[1] *Automatically Tuned Linear Algebra Software (ATLAS).*
URL <http://math-atlas.sourceforge.net/>

[2] Anderson, E.; Bai, Z.; Bischof, C.; aj.: *LAPACK Users‘ guide*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, třetí vydání, 1999, ISBN 0-89871-447-8.
URL <http://www.netlib.org/lapack/lug/lapack_lug.html>

[3] Barrett, R.; Berry, M.; Chan, T. F.; aj.: *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. Philadelphia, PA: SIAM, 1994.
URL <http://www.netlib.org/linalg/html_templates/Templates.html>

[4] Blackford, L. S.; Demmel, J.; Dongarra, J.; aj.: An Updated Set of Basic Linear Algebra Subprograms (BLAS). *ACM Transactions on Mathematical Software*, ročník 28, 2001: s. 135–151.

[5] George, A.; Liu, J. W.: *Computer Solution of Large Sparse Positive Definite*. Prentice Hall Professional Technical Reference, 1981, ISBN 0131652745.

[6] IBM: *Engineering and Scientific Subroutine Library (ESSL).*
URL <http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/topic/com.ibm.cluster.essl44.guideref.doc/am501_dos.html>

[7] Intel: *Sparse Matrix Storage Formats.*
URL <http://www.intel.com/software/products/mkl/docs/webhelp/appendices/mkl_appA_SMSF.html>

[8] Netlib: *Netlib FAQ.*
URL <http://www.netlib.org/misc/faq.html>

[9] Stachoň, M.: *Objektově orientovaná implementace skyline systému ukládání řídkých matic*. 2008.
URL <http://am.vsb.cz/theses/bakalari/2008/pdfs/sta545.pdf>

[10] VŠB-TU Ostrava, Department of Applied Mathematics: *OOSol*.
URL <http://www.am.vsb.cz/oosol/>

[11] Wikipedia, the free encyclopedia: *Basic Linear Algebra Subprograms*.
URL <http://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms>

[12] Wikipedia, the free encyclopedia: *Skyline matrix*.
URL <http://en.wikipedia.org/wiki/Skyline_matrix>

Příloha A

OOSspSkylineStatic.h

Zde jsou uvedeny mnou implementované procedury pro Skyline systém.

```
/*************************************************************************/
/*                                                                       */
/*   OOSol2 2009                                                         */
/*   Dept. of Applied Mathematics,                                       */
/*   Technical University in Ostrava, Czech Rep.                         */
/*                                                                       */
/*   Filename: OOSspSkylineStatic.h                                      */
/*   Created by Radek Pindora, 2009                                      */
/*                                                                       */
/*   List of updates:                                                    */
/*   Date   Author      Description                                      */
/*                                                                       */
/*************************************************************************/
#ifndef OOS_SKYLINE_STATIC_H
#define OOS_SKYLINE_STATIC_H

#include "OOSexceptions.h"
#include "OOSmatrices.h"
#include "OOSmatricesSparseStatic.h"

namespace oosol
{
    class OOSspSkylineStatic: OOSspMatrix
    {

    private:

        bool binary;
        TIND IOFormat;
        double n_one();
        double n_infinity();
```

```
32          double n_max();
            double n_frobenius();
34          /* x <- alfa * A * x */
            void mat_vec(double alpha,OOSspSkylineStatic *A, OOSvectorD *x);
36
        public:
38          TIND row;
            TIND col;
40          double *diag;
            double *uc, *lr;
42          TIND *iuc, *ilr;
            TIND *colProfile, *rowProfile;
44
            TIND size_diagonal, size_uc,size_lr,size_iuc,size_ilr,elements;
46
48          /*****************************************************************/
            /*                          CONSTRUCTORS                         */
50          /*****************************************************************/

52          /* Constructor - initates matrix */
            OOSspSkylineStatic(): OOSspMatrix()
54          {
                init();
56          }

58          /* Constructor - initiates new matrix row*col */
            OOSspSkylineStatic(TIND row, TIND col)
60          {
                resize(row,col);
62              elements = 0;
                double *prvky = new double[elements];
64              TIND *prvkyCol = new TIND[elements];
                TIND *prvkyRow = new TIND[elements];
66              spconvert(prvkyCol,prvkyRow,prvky);
            }
68
            /* Constructor - initiates new matrix */
70          OOSspSkylineStatic(TIND size)
            {
72
                resize(size);
74              elements = 0;
                double *prvky = new double[elements];
76              TIND *prvkyCol = new TIND[elements];
                TIND *prvkyRow = new TIND[elements];
78              spconvert(prvkyCol,prvkyRow,prvky);
            }
80
            /* Constructor - initiates new matrix from 3 vectors
82          col - x coordinates
            row - y coordinates
84          val - value */
```

```
        OOSspSkylineStatic(TIND *row, TIND *col, double *val)
86      {
            init();
88          spconvert(col,row,val);
        }
90
        /** DESTRUCTORY **/
92
        /* Destructor - delete all memebers */
94      virtual ~OOSspSkylineStatic()
        {
96          clean();
        }
98


100     /*************************************************************/
        /*                          METHODS                          */
102     /*************************************************************/

104     /* CLEAN delete all memebers */
        virtual void clean();
106
        /* INIT clean matrix and initiates all to null */
108     virtual void init();

110     /* CONVERT create matrix from 3 vectors
        col - x coordinates
112     row - y coordinates
        val - value */
114     virtual void spconvert(TIND *prvkyCol, TIND *prvkyRow, double *prvky);

116     /* TRANSPOSE transpose this matrix */
        virtual void trans();
118
        /* RESIZE clean matrix and resize to the square */
120     virtual void resize(TIND dim);
        /* RESIZE clean matrix and resize to row x col */
122     virtual void resize(TIND nrow, TIND ncol);


124
        /*************************************************************/
126     /*                          GET/SET                          */
        /*************************************************************/
128
        /* isBinary return true, if IO format is binary */
130     virtual bool isBinary()
        {
132         return binary;
        }
134     /* setBinary set OI format - ascii or binary */
        virtual void setBinary()
136     {
            binary = true;
```

```
138         }
            /* isText return true, if IO format is plain text */
140         virtual bool isText()
            {
142             return !binary;
            }
144         /* setText set OI format - ascii or binary */
            virtual void setText()
146         {
                binary = false;
148         }
            /* getCols return size of matrix - number of Cols */
150         virtual int getCols()
            {
152             return col;
            }
154         /* getRows return size of matrix - number of Rows */
            virtual int getRows()
156         {
                return row;
158         }
            /* nnz return number of non zero numbers */
160         TIND nnz();

162         /* getDiagonal return diagonal element at position i */
            double getDiagonal(TIND i)
164         {
                return diag[i];
166         }
            /* Output/Input format
168         /* 0 = Skyline format
            /* 1 = Sparse matrix format */
170         void setIOFormat(TIND i)
            {
172             IOFormat = i;
            }
174         TIND getIOFormat()
            {
176             return IOFormat;
            }
178
            /*columnHeight return profile of column*/
180         TIND columnHeight(TIND col);
            /*rowHeight return profile of row*/
182         TIND rowHeight(TIND row);

184         /* get return value at row x col */
            virtual double get(TIND row, TIND col);
186         /* get set value at row x col */
            virtual void set(TIND row, TIND col, double val);
188
            /* toString generate MATLAB string output from matrix */
190         virtual string toString();
```

```
192         /* toDenseMatrix create dense matrix from sparse matrix */
            OOSspMatrixStatic* toDenseMatrix();
194
            /*********************************************************************/
196         /*                          MATRIX NORMS                             */
            /*********************************************************************/
198
200         /* norm_1 return 1-norm */
            double norm_1()
202         {
                return n_one();
204         }
            /* norm_inf return infinity norm */
206         double norm_inf()
            {
208             return n_infinity();
            }
210         /* norm_max return max norm */
            double norm_max()
212         {
                return n_max();
214         }
            /*norm_F return Frobenius norm*/
216         double norm_F()
            {
218             return n_frobenius();
            }
220
            /*********************************************************************/
222         /*                  MATRIX-VECTOR OPERATIONS                         */
            /*********************************************************************/
224
            /* mv compute matrix-vector product
226         /* x<-alpha * A * x + beta * y */
            void mv(OOSvectorD *x, OOSvectorD *y, double alpha, double beta);
228
            /* mtv compute matrix-vector product */
230         /* x<-alpha * A^T * x + beta * y */
            void mtv(OOSvectorD *x, OOSvectorD *y, double alpha, double beta);
232
            /* sum_mv compute summed matrix-vector multiplies */
234         /* y <- alpha*A*x + beta*B*x */
            void sum_mv(OOSspSkylineStatic *B, OOSvectorD *x, OOSvectorD *y,
236             double alpha, double beta);

238         /* mvt compute multiple matrix vector multiplies */
            /* x <- beta * A^T * y +z
240         /* w <- alpha * A * x */
            void mvt(OOSvectorD *z, OOSvectorD *y, OOSvectorD *w, OOSvectorD *x,
242             double alpha, double beta);
```

```
244         /* mver compute multiple matrix vector mults and low rank updates */
            /* A <- A + u1*v1^T + u2*v2^T  */
246         /* x <- beta * A^T * y + z */
            /* w <- aplha * A * x */
248         void mver(OOSvectorD *z, OOSvectorD *y, OOSvectorD *x,
                OOSvectorD *w, OOSvectorD *u1, OOSvectorD *u2,
250             OOSvectorD *v1, OOSvectorD *v2, double alpha,
                double beta);
252
            /* r compute rank one updates */
254         /*A <- alpha * x * y^T + beta * A*/
            void r(OOSvectorD *x, OOSvectorD *y,
256             double alpha, double beta);

258         /* syr2 compute rank one&two updae
            A <- (alpha * x) * y^T + y * (alpha * x)^T + beta * A  */
260         void syr2(OOSvectorD *x, OOSvectorD *y,
                double alpha, double beta);
262

264     }; //end of OOSspSkylineStatic


266
        /*********************************************************************/
268     /*                           STREAMS                                 */
        /*********************************************************************/
270
        /** Overloaded output stream operator */
272     ostream& operator<<(ostream& os, OOSspSkylineStatic& mat);
        /** Overloaded input file stream operator */
274     ifstream& operator>>(ifstream& is, OOSspSkylineStatic& mat);
        /** Overloaded output file stream operator */
276     ofstream& operator<<(ofstream& os, OOSspSkylineStatic& mat);


278
    } //end of namespace oosol
280

282 #endif /* OOS_SKYLINE_STATIC_H */
```

Příloha B

OOSspSkylineStatic.cpp

```
/*************************************************************************/
/*                                                                       */
/*   OOSol2 2009                                                         */
/*   Dept. of Applied Mathematics,                                       */
/*   Technical University in Ostrava, Czech Rep.                         */
/*                                                                       */
/*   Filename: OOSspSkylineStatic.cpp                                    */
/*   Created by Radek Pindora, 4.2009                                    */
/*                                                                       */
/*   List of updates:                                                    */
/*   Date   Author      Description                                      */
/*                                                                       */
/*************************************************************************/

#include "OOSspSkylineStatic.h"

namespace oosol
{

    /*************************************************************************/
    /*                                                                       */
    /*  Method definitions of class OOSspSkylineStatic                       */
    /*                                                                       */
    /*************************************************************************/


    /*************************************************************************/
    /*                            METHODS                                    */
    /*************************************************************************/

    void OOSspSkylineStatic::clean()
    {
        delete[] diag;
        delete[] uc;
```

```
        delete[] lr;
        delete[] iuc;
        delete[] ilr;
        delete[] colProfile;
        delete[] rowProfile;
    }

    void OOSspSkylineStatic::init()
    {
        diag = NULL;
        uc = NULL;
        lr = NULL;
        iuc = NULL;
        ilr = NULL;
        colProfile = NULL;
        rowProfile = NULL;

        IOFormat = 1;
        col = 0;
        row = 0;
        binary = false;
        size_diagonal = 0;
        size_uc = 0;
        size_lr = 0;
        size_iuc = 0;
        size_ilr = 0;
    }

    void OOSspSkylineStatic::resize(TIND dim)
    {
        init();
        ASSERT(dim>=0);
        row = dim; col = dim;
    }

    void OOSspSkylineStatic::resize(TIND nrow,TIND ncol)
    {
        init();
        ASSERT(nrow>=0 && ncol>=0);
        row = nrow; col = ncol;
    }

    void OOSspSkylineStatic::trans()
    {
        TIND *tTIND;
        double *tDouble;
        TIND temp;

        tTIND = colProfile;
        colProfile = rowProfile;
        rowProfile = tTIND;

        temp = row;
```

```
88          row = col;
            col = temp;
90
            tDouble = uc;
92          uc = lr;
            lr = tDouble;
94
            tTIND = iuc;
96          iuc = ilr;
            ilr = tTIND;
98
            temp = size_uc;
100         size_uc = size_lr;
            size_lr = temp;
102
            temp = size_iuc;
104         size_iuc = size_ilr;
            size_ilr = temp;
106     }

108     TIND OOSspSkylineStatic::nnz()
        {
110         elements = 0;
            for(TIND i = 0; i < size_uc; i++)
112             if(uc[i] != 0) elements++;
            for(TIND i = 0; i < size_lr; i++)
114             if(lr[i] != 0) elements++;
            for(TIND i = 0; i < size_diagonal; i++)
116             if(diag[i] != 0) elements++;
            return elements;
118     }
        string OOSspSkylineStatic::toString()
120     {
            std::stringstream o;
122         o.precision(10);

124         for(int i = 0; i < row; i++)
            {
126
                for(int y = 0; y < col; y++)
128             {
                    TIND oy = y+1, oi = i+1;
130                 double val = this->get(i,y);
                    if(val != 0)
132                 {
                        TIND oy = y+1, oi = i+1;
134                     double val = this->get(i,y);
                        if(val != 0) o << oi << " "
136                         << oy << " " << val << endl;
                    }
138             }
            }
140         return o.str();
```

```
    }

OOSspMatrixStatic* OOSspSkylineStatic::toDenseMatrix()
{
    OOSspMatrixStatic *out = new OOSspMatrixStatic(row,col);
    for(TIND r = 0; r < row; r++)
    {
        for(TIND c = 0; c < col; c++)
        {
            out->set(r,c,this->get(r,c));
        }
    }
    return out;
}



void OOSspSkylineStatic::spconvert(TIND *prvkyRow, TIND *prvkyCol, double *prvky)
{
    TIND aun = 0; //count of elements in AU
    TIND aln = 0; //count of elements in AL
    size_diagonal = MIN(col,row);
    diag = new double[size_diagonal];
    colProfile = new TIND[col];
    rowProfile = new TIND [row];


    std::fill(diag,diag+MIN(col,row),0.0);
    TIND elements_new = 0;
    for(int i = 0; i <col; i++) colProfile[i] = i;
    for(int i = 0; i <row; i++) rowProfile[i] = i;

    for(int i = 0; i < elements; i++)
    {
        TIND x = prvkyCol[i];
        TIND y = prvkyRow[i];
        double val = prvky[i];

        if(x == y) //diagonal elements
        {
            diag[x] = val; //save to diagnoal vector
        }
        else
        {
            prvky[elements_new] = val;
            prvkyCol[elements_new] = x;
            prvkyRow[elements_new] = y;
            elements_new++;
            if(y < x) //upper triangular matrix
            {
                //profile above the diagonal
                if(colProfile[x]== x) colProfile[x] = y;
                else if(colProfile[x] > y) colProfile[x] = y;
```

```
194                 }
                    else if(y > x) //lower triangular matrix
196                 {
                        //profile under the diagonal
198                     if(rowProfile[y]== y) rowProfile[y] = x;
                        else if(rowProfile[y] > x) rowProfile[y] = x;
200                 }
                }
202         }

204         size_iuc = col+1;
            iuc = new TIND[size_iuc];
206         iuc[0] = -1;
            iuc[1] = 0;
208         TIND x = 0;

210         //col profile
            for(int i = 1; i < col; i++)
212         {

214             if(colProfile[i] != i) colProfile[i] = i - colProfile[i];
                else colProfile[i] = 0;
216             //cout << " " << colProfile[i];
                aun +=colProfile[i];
218
                if(colProfile[i] == 0 ) iuc[i] = iuc[i-1];
220             else
                {
222
                    iuc[i] = x;
224                 x += colProfile[i];
                }
226         }
            iuc[size_iuc-1] = x;
228



230
            size_ilr = row+1;
232         ilr = new TIND[size_ilr];
            ilr[0] = -1;
234         ilr[1] = 0;
            TIND y = 0;
236
            //row profile
238         for(int i = 1; i < row; i++)
            {
240
                if(rowProfile[i] != i) rowProfile[i] = i - rowProfile[i];
242             else rowProfile[i] = 0;
                //cout << " " << colProfile[i];
244             aln +=rowProfile[i];

246             if(rowProfile[i] == 0 ) ilr[i] = ilr[i-1];
```

```
            else
            {

                ilr[i] = y;
                y += rowProfile[i];
            }
        }
        ilr[size_ilr-1] = y;


        size_uc = aun;
        uc = new double[size_uc];
        std::fill(uc,uc+aun,0.0);
        size_lr = aln;
        lr = new double[size_lr];
        std::fill(lr,lr+aln,0.0);



        TIND x1;
        for(int i = 0; i < elements; i++)
        {
            if(prvkyRow[i] < prvkyCol[i]) //above the diagonal
            {
                x1 = iuc[prvkyCol[i]] + (prvkyCol[i]-prvkyRow[i] - 1);
                uc[x1] = prvky[i];
            }
            else if(prvkyRow[i] > prvkyCol[i]) //under the diagonal
            {
                x1 = ilr[prvkyRow[i]] + (prvkyRow[i]-prvkyCol[i] - 1);
                lr[x1] = prvky[i];
            }
        }

    }


    /*****************************************************************/
    /*                            GET/SET                            */
    /*****************************************************************/


    TIND OOSspSkylineStatic::columnHeight(TIND col)
    {
        return colProfile[col];

    }
    TIND OOSspSkylineStatic::rowHeight(TIND row)
    {
        return rowProfile[row];
    }

    double OOSspSkylineStatic::get(TIND row,TIND col)
```

```
300     {
            if(col == row) return diag[col];
302         if(row < col) //above the diagonal
            {
304             TIND profil = col - row;
                if(profil > columnHeight(col)) return 0;
306             return uc[iuc[col]+profil-1];

308         }
            if(row > col) //under the diagonal
310         {
                TIND profil = row - col;
312             if(profil > rowHeight(row)) return 0;
                return lr[ilr[row]+profil-1];
314         }
            return 0;
316     }

318     void OOSspSkylineStatic::set(TIND row,TIND col, double val)
        {
320
            if(col == row) diag[col] = val;
322         if(row < col) //above the diagonal
            {
324
                TIND profil = col - row;
326             if(profil <= columnHeight(col)) uc[iuc[col]+profil-1] = val;
                else if(val != 0)
328             {

330                 TIND profilDif = profil - columnHeight(col);
                    double *AUnew = new double[size_uc+profilDif];
332                 std::fill(AUnew,AUnew+(size_uc+profilDif),0.0);

334                 TIND IDUold;
                    TIND i = col+1;
336                 while(columnHeight(i) == 0) i++;
                    IDUold = iuc[i] - 1;
338                 if(columnHeight(col) == 0) iuc[col] = iuc[i];

340                 for(TIND i = col+1; i < size_iuc; i++)
                    {
342                     if(columnHeight(i) == 0) continue;
                        iuc[i] += profilDif;
344                 }
                    //copy to bigger array
346                 int x = 0;
                    for(x; x <= IDUold; x++)
348                 {
                        AUnew[x] = uc[x];
350                 }
                    for(x; x < size_uc; x++)
352                 {
```

```
                AUnew[x+profilDif] = uc[x];
            }

            delete[] uc;
            uc = AUnew;
            size_uc += profilDif;
            if(colProfile[col] != -1) colProfile[col] += profilDif;
            else colProfile[col] = profilDif;
            uc[iuc[col]+profil-1] = val;

        }
    }
    if(row > col) //under the diagonal
    {
        TIND profil = row - col;
        if(profil <= rowHeight(row)) lr[ilr[row]+profil-1] = val;
        else if(val != 0)
        {
            TIND profilDif = profil - rowHeight(row);
            double *ALnew = new double[size_lr+profilDif];
            std::fill(ALnew,ALnew+(size_lr+profilDif),0.0);

            TIND IDLold;
            TIND i = row+1;
            while((rowHeight(i) == 0))  i++;
            IDLold = ilr[i] - 1;
            //if (IDLold == -1) IDLold = 0;
            if(rowHeight(row) == 0) ilr[row] = ilr[i];

            for(TIND i = row+1; i < size_ilr; i++)
            {
                if(rowHeight(i) == 0) continue;
                ilr[i] += profilDif;
            }
            //copy to bigger array
            int x = 0;
            for(x; x <= IDLold; x++)
            {
                ALnew[x] = lr[x];
            }
            for(x; x < size_lr; x++)
            {
                ALnew[x+profilDif] = lr[x];
            }
            //delete[] lr;
            lr = ALnew;
            size_lr += profilDif;
            if(rowProfile[row] != -1) rowProfile[row] += profilDif;
            else rowProfile[row] = profilDif;
            lr[ilr[row]+profil-1] = val;
        }
    }
}
```

```
406
408     /*****************************************************************/
        /*                            STREAMS                            */
410     /*****************************************************************/

412

414     /** Overloaded input file stream operator */
        ifstream& operator>>(ifstream& is, OOSspSkylineStatic& mat)
416     {
            TIND col, row, elements;
418         double *prvky;
            TIND *prvkyCol, *prvkyRow;
420
            if(mat.getIOFormat() == 0)
422         {
                is >> mat.col;
424             is >> mat.row;
                is >> mat.elements;
426             is >> mat.size_uc;
                is >> mat.size_iuc;
428             is >> mat.size_lr;
                is >> mat.size_ilr;
430             mat.size_diagonal = MIN(mat.col,mat.row);

432             mat.uc = new double[mat.size_uc];
                mat.iuc = new TIND[mat.size_iuc];
434             mat.lr = new double[mat.size_lr];
                mat.ilr = new TIND[mat.size_ilr];
436             mat.diag = new double[mat.size_diagonal];
                mat.colProfile = new TIND[mat.col];
438             mat.rowProfile = new TIND[mat.row];

440             for(int i = 0; i < mat.size_diagonal; i++) is >> mat.diag[i];
                for(int i = 0; i < mat.size_uc; i++) is >> mat.uc[i];
442             for(int i = 0; i < mat.size_iuc; i++) is >> mat.iuc[i];
                for(int i = 0; i < mat.size_lr; i++) is >> mat.lr[i];
444             for(int i = 0; i < mat.size_ilr; i++) is >> mat.ilr[i];
                for(int i = 0; i < mat.col; i++) is >> mat.colProfile[i];
446             for(int i = 0; i < mat.row; i++) is >> mat.rowProfile[i];
                return is;
448         }

450         if(mat.isText())
            {
452             ASSERT(is.good());
                is >> col;
454             ASSERT(is.good());
                is >> row;
456             ASSERT(is.good());
                is >> elements;
458             mat.resize(row,col);
```

```
            mat.elements = elements;
460
            prvky = new double[elements];
462         prvkyCol = new TIND[elements];
            prvkyRow = new TIND[elements];
464         double c, r;

466         for(int i = 0; i < elements; i++)
            {
468
                ASSERT(is.good());
470             is >> r;
                ASSERT(is.good());
472             is >> c;
                ASSERT(is.good());
474             prvkyCol[i] = (TIND)c;
                prvkyRow[i] = (TIND)r;
476
                is >> prvky[i];
478
                prvkyCol[i] -= 1;
480             prvkyRow[i] -= 1;
            }
482     }
        else
484     {
            ASSERT(is.good());
486         is.read((char*)&col,sizeof(TIND));
            ASSERT(is.good());
488         is.read((char*)&row,sizeof(TIND));
            ASSERT(is.good());
490         is.read((char*)&elements,sizeof(TIND));
            mat.resize(row,col);
492         mat.elements = elements;

494         prvky = new double[elements];
            prvkyCol = new TIND[elements];
496         prvkyRow = new TIND[elements];

498         for(int i = 0; i < elements; i++)
            {
500             ASSERT(is.good());
                is.read((char*)&prvkyRow[i],sizeof(TIND));
502             ASSERT(is.good());
                is.read((char*)&prvkyCol[i],sizeof(TIND));
504             ASSERT(is.good());
                is.read((char*)&prvky[i],sizeof(double));
506
                prvkyCol[i] -= 1;
508             prvkyRow[i] -= 1;
            }
510
```

```
512         }
            mat.spconvert(prvkyRow,prvkyCol,prvky);
514         return is;
        }
516     /** Overloaded output file stream operator */
        ofstream& operator<<(ofstream& os, OOSspSkylineStatic& mat)
518     {
            if(mat.getIOFormat() == 0)
520         {
                os << mat.col << " " << mat.row << " " << mat.nnz() << endl;
522             os << mat.size_uc << " " << mat.size_iuc
                    << " " << mat.size_lr << " " << mat.size_ilr << endl;
524
                os << endl;
526             for(int i = 0; i < mat.size_diagonal; i++) os
                    << mat.diag[i] << " ";
528             os << endl;
                for(int i = 0; i < mat.size_uc; i++) os
530                 << mat.uc[i] << " ";
                os << endl;
532             for(int i = 0; i < mat.size_iuc; i++) os
                    << mat.iuc[i] << " ";
534             os << endl;
                for(int i = 0; i < mat.size_lr; i++) os
536                 << mat.lr[i] << " ";
                os << endl;
538             for(int i = 0; i < mat.size_ilr; i++) os
                    << mat.ilr[i] << " ";
540             os << endl;
                for(int i = 0; i < mat.col; i++)
542                 os << mat.columnHeight(i) << " ";
                os << endl;
544             for(int i = 0; i < mat.row; i++)
                    os << mat.rowHeight(i) << " ";
546         }
            if(mat.getIOFormat() == 1)
548         {
                if(mat.isBinary())
550             {
                    TIND el = mat.nnz();
552                 os.write((const char*)&mat.col,sizeof(TIND));
                    os.write((const char*)&mat.row,sizeof(TIND));
554                 os.write((const char*)&el,sizeof(TIND));
                    for(int i = 0; i < mat.row; i++)
556                 {
                        for(int y = 0; y < mat.col; y++)
558                     {
                            double val = mat.get(i,y);
560                         if(val == 0) continue;
                            TIND oy = y+1, oi = i+1;
562                         os.write((const char*)&oi,sizeof(TIND));
                            os.write((const char*)&oy,sizeof(TIND));
564                         os.write((const char*)&val,sizeof(double));
```

```
566                     }
                    }
568             }
                else
570             {
                    os << mat.col << " " << mat.row << " " << mat.nnz() << endl;
572                 for(int i = 0; i < mat.row; i++)
                    {
574                     for(int y = 0; y < mat.col; y++)
                        {
576                         TIND oy = y+1, oi = i+1;
                            double val = mat.get(i,y);
578                         if(val != 0) os << oi << " " << oy
                                << " " << val << endl;
580                     }
                    }
582             }
        }
584     return os;
    }
586
    /** Overloaded output stream operator */
588 ostream& operator<<(ostream& os, OOSspSkylineStatic& mat)
    {
590
        os << endl << "\nTestovaci vypis Matice: \n" << "Col: " << mat.getCols();
592     os << " Rows: " << mat.getRows();
        os << " Elements: "<< mat.nnz();
594
        cout << "\n\n Sloupcovy profil: ";
596     for(int i = 0; i < mat.col; i++)
        {
598         cout << " " << mat.columnHeight(i);
        }
600     cout << "\n Radkovy profil: ";
        for(int i = 0; i < mat.row; i++)
602     {
            cout << " " << mat.rowHeight(i);
604     }

606     cout << endl <<"\nPrvky na diagonale: ";
        for(int i = 0; i < mat.size_diagonal; i++)
608     {
            cout << " " << mat.getDiagonal(i);
610     }
        cout <<endl<<"\nIDU: ";
612     for(int i = 0; i < mat.size_iuc; i++)
        {
614         cout<<mat.iuc[i]<<" ";
        }
616     cout <<endl<<"AU: ";
        for(int i = 0; i < mat.size_uc; i++)
```

```
618         {
                cout<<mat.uc[i]<<" ";
620         }

622         cout <<endl<<"\nIDL: ";
            for(int i = 0; i < mat.size_ilr; i++)
624         {
                cout<<mat.ilr[i]<<" ";
626         }
            cout <<endl<<"AL: ";
628         for(int i = 0; i < mat.size_lr; i++)
            {
630             cout<<mat.lr[i]<<" ";
            }
632         cout << endl;

634         //TODO
            return os;
636     }

638 } //end of namespace
```

Příloha C

OOSspSkylineStaticBLAS.cpp

```
/*************************************************************************/
/*                                                                       */
/*   OOSol2 2009                                                         */
/*   Dept. of Applied Mathematics,                                       */
/*   Technical University in Ostrava, Czech Rep.                         */
/*                                                                       */
/*   Filename: OOSspSkylineStaticBLAS.cpp                                */
/*   Created by Radek Pindora, 4.2009                                    */
/*                                                                       */
/*   List of updates:                                                    */
/*   Date   Author      Description                                      */
/*                                                                       */
/*************************************************************************/

#include "OOSspSkylineStatic.h"

namespace oosol
{
    /*************************************************************************/
    /*                    MATRIX-VECTOR OPERATIONS                           */
    /*************************************************************************/

    void OOSspSkylineStatic::mat_vec(double alpha,
        OOSspSkylineStatic *A,
        OOSvectorD *x)
    {
        TIND row = A->getRows(), col = A->getCols();
        OOSvectorD *sum = new OOSvectorD(row,0.0);
        double ab = 0.0;

        if ((A->getCols() != x->getAllocDim()))
            throw new OOSdimensionException();
        if(alpha == 0)
        {
```

```
            x->resize(sum->getAllocDim());
            x->setVal(sum->getVal());
        }
        else
        {
            for(TIND r = 0; r < row; r++) //every row
            {
                if( r <= col)
                {

                    for(TIND i = r; i < col; i++) //right&diagonal
                    {
                        ab = A->get(r,i)*x->get(i)*alpha;

                        sum->set(r,sum->get(r)+ab);
                    }

                    TIND rProf = A->rowHeight(r);
                    TIND cStart = r - rProf;
                    TIND auStart = A->ilr[r];
                    if(rProf != 0)
                    {

                        for(TIND i = 0; i < rProf; i++)
                        {
                            ab = A->lr[auStart-i+rProf-1]*x->get(i+cStart)*alpha;
                            sum->set(r,sum->get(r)+ab);
                        }
                    }
                }
                else
                {
                    TIND rProf = A->rowHeight(r);
                    TIND cStart = col - rProf+1;
                    TIND auStart = A->ilr[r];
                    if(rProf != 0)
                    {

                        for(TIND i = 0; i < rProf-1; i++)
                        {
                            double a = A->lr[auStart-i+rProf-1];
                            double b = x->get(i+cStart);
                            ab = a*b*alpha;
                            sum->set(r,sum->get(r)+ab);
                        }
                    }
                }
            }
            //x->resize(row);
            //x->setVal(sum->getVal());
            x->clean();
            x->copy(*sum);
```

```
88
        }
90  }

92
    void OOSspSkylineStatic::mv(OOSvectorD *x, OOSvectorD *y, double alpha,
94      double beta)
    {
96      if ((this->getCols() != x->getAllocDim())
            || (this->getRows() != y->getAllocDim()))
98          throw new OOSdimensionException();

100     OOSvectorD *v1 = new OOSvectorD(*x);
        mat_vec(alpha,this,v1);
102     for(TIND i = 0; i < y->getAllocDim(); i++)
            y->set(i,v1->get(i)+(y->get(i)*beta));
104 }

106
    void OOSspSkylineStatic::mtv(OOSvectorD *x, OOSvectorD *y, double alpha,
108     double beta)
    {
110     this->trans();
        if ((this->getCols() != x->getAllocDim())
112         || (this->getRows() != y->getAllocDim()))
            throw new OOSdimensionException();
114
        OOSvectorD *v1 = new OOSvectorD(*x);
116     mat_vec(alpha,this,v1);
        for(TIND i = 0; i < y->getAllocDim(); i++)
118         y->set(i,v1->get(i)+(y->get(i)*beta));
        this->trans();
120 }

122
    void OOSspSkylineStatic::sum_mv(OOSspSkylineStatic *B, OOSvectorD *x, OOSvectorD *y,
124     double alpha, double beta)
    {
126     if ((this->getCols() != x->getAllocDim())
            || (B->getCols() != x->getAllocDim())
128         || (this->getRows() != B->getRows()))
            throw new OOSdimensionException();
130     OOSvectorD *v1 = new OOSvectorD(*x);
        OOSvectorD *v2 = new OOSvectorD(*y);
132     OOSvectorD *out = new OOSvectorD(*y);

134     mat_vec(alpha,this,v1);
        mat_vec(beta,B,v2);
136
        if (v1->getAllocDim() != v2->getAllocDim()) throw new OOSdimensionException();
138

140     y->resize(v1->getAllocDim());
```

```
        for(TIND i = 0; i < v1->getAllocDim(); i++)
142     {   double a = v1->get(i);
        double b = v2->get(i);
144     y->set(i,a+b);
        }
146
    }
148

150 void OOSspSkylineStatic::mvt(OOSvectorD *z, OOSvectorD *y, OOSvectorD *w, OOSvectorD *x,
        double alpha, double beta)
152 {
        this->trans();
154     if ((this->getCols() != y->getAllocDim()))
            throw new OOSdimensionException();
156
        OOSvectorD *v1 = new OOSvectorD(*y);
158     mat_vec(beta,this,v1);

160     if ((z->getAllocDim() != v1->getAllocDim()))
            throw new OOSdimensionException();
162
        x->resize(v1->getAllocDim());
164     for(int i = 0; i < v1->getAllocDim(); i++) x->set(i,v1->get(i) + z->get(i));

166     this->trans();

168     OOSvectorD *v2 = new OOSvectorD(*x);
        mat_vec(alpha,this,v2);
170
        w->resize(v2->getAllocDim());
172     w->setVal(v2->getVal());
    }
174

176 void OOSspSkylineStatic::mver(OOSvectorD *z, OOSvectorD *y, OOSvectorD *x,
        OOSvectorD *w, OOSvectorD *u1, OOSvectorD *u2,
178     OOSvectorD *v1, OOSvectorD *v2, double alpha,
        double beta)
180 {
        if ((this->getRows() != u1->getAllocDim()) ||
182         (this->getCols() != v1->getAllocDim()) )
            throw new OOSdimensionException();
184     if ((u2->getAllocDim() != u1->getAllocDim()) ||
            (v2->getAllocDim() != v1->getAllocDim()) )
186         throw new OOSdimensionException();

188     TIND rows = this->getRows(), cols = this->getCols();
        double val = 0.0;
190
        for(TIND k = 0; k < MIN(rows,cols); k++)
192     {
            for(TIND r = k; r < rows; r++) //dolu
```

```
194             {
                    val = u1->get(r) * v1->get(k)+u2->get(r) * v2->get(k);
196                 if(val != 0) this->set(r,k,val+this->get(r,k));
                }
198             for(TIND c = k+1; c < cols; c++) //vpravo
                {
200                 val = u1->get(k) * v1->get(c) + u2->get(k) * v2->get(c);
                    if(val != 0) this->set(k,c,val+this->get(k,c));
202
                }
204         }
            mvt(z,y,w,x,alpha,beta);
206
    }
208     void OOSspSkylineStatic::r(OOSvectorD *x, OOSvectorD *y,
            double alpha, double beta)
210     {
            if ((this->getRows() != x->getAllocDim()) || (this->getCols() != y->getAllocDim()) )
212             throw new OOSdimensionException();
            TIND rows = this->getRows(), cols = this->getCols();
214         double val = 0.0;

216         for(TIND k = 0; k < MIN(rows,cols); k++)
            {
218             for(TIND r = k; r < rows; r++) //dolu
                {
220                 val = x->get(r) * y->get(k) * alpha;
                    if(val != 0) this->set(r,k,val+this->get(r,k)*beta);
222             }
                for(TIND c = k+1; c < cols; c++) //vpravo
224             {
                    val = x->get(k) * y->get(c) * alpha;
226                 if(val != 0) this->set(k,c,val+this->get(k,c)*beta);
                }
228         }
    }
230
        void OOSspSkylineStatic::syr2(OOSvectorD *x, OOSvectorD *y,
232         double alpha, double beta)
    {
234         TIND rows = this->getRows(), cols = this->getCols();
        if ( (rows != cols) || (x->getAllocDim() != y->getAllocDim())
236         || (rows != x->getAllocDim()) )
            throw new OOSdimensionException();
238
        double val = 0.0;
240
        for(TIND k = 0; k < MIN(rows,cols); k++)
242     {
            for(TIND r = k; r < rows; r++) //dolu
244         {
                val=((alpha*x->get(r))*y->get(k)) + (x->get(k)*(alpha*y->get(r)));
246             if(val != 0) this->set(r,k,val+this->get(r,k)*beta);
```

```
            }
            for(TIND c = k+1; c < cols; c++) //vpravo
            {
                val = ((alpha*x->get(k))*y->get(c)) + (x->get(c)*(alpha*y->get(k)));
                if(val != 0) this->set(k,c,val+this->get(k,c)*beta);
            }
        }
    }


    /*****************************************************************/
    /*                          MATRIX NORMS                         */
    /*****************************************************************/

    double OOSspSkylineStatic::n_one() //sloupcova
    {
        TIND row = this->row, col = this->col;
        double max = 0;
        double *sum = new double[col];
        std::fill(sum,sum+col,0.0);

        for(TIND i = 0; i < this->size_diagonal; i++) //prvky na diagonale
        {
            sum[i] = ABS(this->diag[i]);
        }
        for(TIND c = 0; c < col; c++)
        {
            if(this->iuc[c] != -1) //prvky nad diagonalou
            {
                TIND u = this->iuc[c];
                TIND um = u + this->colProfile[c];
                for(TIND i = u; i < um; i++)
                {
                    sum[c] += ABS(this->uc[i]);
                }
            }
            for(TIND i = c+1; i < row; i++) //prvky pod diagonalou
            {
                sum[c] += ABS(get(i,c));
            }

        }
        for(TIND i = 0; i < col; i++) max = MAX(max,sum[i]);
        return max;
    }

    double OOSspSkylineStatic::n_infinity() //radkova
    {
        TIND row = this->row, col = this->col;
        double max = 0;
        double *sum = new double[row];
        std::fill(sum,sum+row,0.0);
```

```
300
        for(TIND i = 0; i < this->size_diagonal; i++) //prvky na diagonale
302     {
            sum[i] = ABS(this->diag[i]);
304     }
        for(TIND c = 0; c < row; c++)
306     {
            if(this->ilr[c] != -1) //prvky vlevo
308         {
                TIND u = this->ilr[c];
310             TIND um = u + this->rowProfile[c];
                for(TIND i = u; i < um; i++)
312             {
                    sum[c] += ABS(this->lr[i]);
314             }
            }
316         for(TIND i = c+1; i < col; i++) //prvky vpravo
            {
318             sum[c] += ABS(get(c,i));
            }
320
        }
322     for(TIND i = 0; i < row; i++) max = MAX(max,sum[i]);
        return max;
324 }

326 double OOSspSkylineStatic::n_max()
    {
328     double max = get(0,0);

330     for(TIND i = 0; i < this->size_uc; i++)
            max = MAX(max,ABS(this->uc[i]));
332     for(TIND i = 0; i < this->size_lr; i++)
            max = MAX(max,ABS(this->lr[i]));
334     for(TIND i = 0; i < this->size_diagonal; i++)
            max = MAX(max,ABS(this->diag[i]));
336
        return max;
338 }


340
    double OOSspSkylineStatic::n_frobenius()
342 {
        double max = 0;
344
        for(TIND i = 0; i < this->size_uc; i++)
346         max += pow(this->uc[i],2);
        for(TIND i = 0; i < this->size_lr; i++)
348         max += pow(this->lr[i],2);
        for(TIND i = 0; i < this->size_diagonal; i++)
350         max += pow(this->diag[i],2);
        return sqrt(max);
352 }
```

```
354
} //end of namespace
```

Příloha D

oosollitetest.cpp

Zde je uvedena testovací třída pro *OOSspSkylineStatic*.

```
#include "OOSvectors.h"
#include "OOSmatrices.h"
#include "OOSspSkylineStatic.h"
#include <iostream>
//#pragma comment(lib,"oosol.lib")

using std::cout;
using std::cin;
using namespace oosol;

int main()
{

    try
    {
        std::cout.precision(10);
            //vytvoreni matic

        OOSspSkylineStatic *A = new OOSspSkylineStatic(10, 10);
        OOSspSkylineStatic *B = new OOSspSkylineStatic(10,10);

            //naplneni matic nahodnymi cisly
        double k = 1;
        for(TIND j = 0; j < A->getRows(); j++)
        for(TIND i = j; i < A->getCols(); i++)
            {
                A->set(j,i,rand());
                B->set(j,i,k);
                k++;
            }

```

```
            //nasleduje testovani blas rutin
33          OOSvectorD *x = new OOSvectorD(A->getCols(),1.0);
            OOSvectorD *y = new OOSvectorD(A->getCols(),1.0);
35          OOSvectorD *z = new OOSvectorD(A->getCols(),1.0);
            OOSvectorD *w = new OOSvectorD(A->getCols(),1.0);
37          for(int i = 0; i < y->getAllocDim(); i++) x->set(i,i);
            for(int i = 0; i < z->getAllocDim(); i++) y->set(i,i*i);
39          OOSvectorD *out = new OOSvectorD(*y);
            double alpha = 2.0;
41          double beta = 3.0;

43          //BLAS normy
            cout << "\nNorm One: " << A->norm_1();
45          cout << "\nNorm Infinity: " << A->norm_inf();
            cout << "\nNorm Max: " << A->norm_max();
47          cout << "\nNorm Frobenius: " << A->norm_F();


49

            //testovani operace sum_mv
51          A->mvt(z,y,w,x,alpha,beta);

53          //ukazka vystupu do souboru test.m
            ofstream m("E:\\out\\test.m");
55          m   << "function[] = test()" << endl
                <<"load A.txt; " << endl
57              <<"A = spconvert(A);" << endl
                <<"load X.txt;" << endl
59              <<"load Y.txt;" << endl
                <<"load Z.txt;" << endl
61              <<"load W.txt;" << endl
                <<"nA1 = norm(A,1)" << endl
63              <<"nAi = norm(A,inf)" << endl
                <<"nAf = norm(A,'fro');" << endl
65              <<"xM = 3 * A' * Y + Z;" << endl
                <<"wM = 2 * A * xM;" << endl
67              << "diff1 = norm(xM - X,1)" << endl
                << "diff2 = norm(wM - W,1)"<<endl;
69

            //postupne vypisy objektu do souboru
71          ofstream o("E:\\out\\A.txt");
            o << A->toString();
73          o.close();
            ofstream ox("E:\\out\\X.txt");
75          string a = x->toString();
            ox << a;
77          ox.close();
            ofstream oy("E:\\out\\Y.txt");
79          string b = y->toString();
            oy << b;
81          oy.close();
            ofstream ou("E:\\out\\Z.txt");
83          string ou1 = z->toString();
            ou << ou1;
```

```
85          ou.close();
            ofstream oiu("E:\\out\\W.txt");
87          string oiu1 = w->toString();
            oiu << oiu1;
89          oiu.close();

91                  //vytvoreni huste matice
            OOSspMatrixStatic *DM = A->toDenseMatrix();
93


95      } catch (OOSexception e)
        {
97          e.print();
            cin.get();
99          return 1;
        }
101     cout << "\n\nEnd...";
        cin.get();
103     return 0;
}
```